Parallel Implementation of a Protein Structure Refinement Algorithm

JOHN R. GUNN*

Département de Chimie, Université de Montréal, C.P. 6128, Succ. Centre-ville, Montréal, Québec H3C 3J7, Canada

RICHARD A. FRIESNER

Department of Chemistry and Center for Biomolecular Simulation, Columbia University, New York, New York 10027

Received 23 January 1995; accepted 21 September 1995

ABSTRACT

An algorithm is presented for the refinement of reduced-model structures of proteins. A simulated annealing minimization is carried out in which the trial moves consist of the replacement of three-residue segments from a presorted library. The segments in the library are screened independently by their net end-to-end rotations so as to reproduce a distribution of conformations in the library similar to those in the ensemble. A general form of a look-up table contact potential is used to evaluate the free energy. This algorithm has been implemented on a parallel connection machine, on which a large number of molecules can be simultaneously simulated. The calculation of the pairwise distance matrix is distributed across the nodes of the machine to achieve an increase in performance and a reduction in the memory required on each node to store the potential table. The results of the refinement are shown for the test case of myoglobin, and the parallel performance is compared to that of a serial version of the algorithm.

Introduction

The problem of sampling the possible conformations of a large flexible molecule like a protein is a formidable challenge for a computer simulation. It is necessary to locate the global minimum of a very high-dimensional potential surface composed of many nearly equal interactions. For even the most sophisticated methods available, the computational capacity of available computers remains the major limitation to the size of protein that can be reasonably studied.

We have developed a hierarchical algorithm

*Author to whom all correspondence should be addressed.

CCC 0192-8651 / 96 / 101217-12

which partially addresses the problem of reducing the scaling of the computation with the complexity of the protein in a method based on a Monte Carlo simulated annealing procedure.[1] This method uses a simplified model of the secondary structure of the protein in which rigid cylinders corresponding to the structural units are connected by spheres representing the flexible loops. This is combined with a crude long-range potential to model the packing of α-helices and β-strands into a hydrophobic core. The simplified representation of the molecule is coupled to a more detailed representation of the individual residues which allows the results of the rapid trial moves of the reduced model to be evaluated with a higher level potential leading to the generation of more accurate results. The result is that most of the calculation depends only on the number of secondary structural segments rather than the number of residues with a corresponding reduction in the complexity of the problem. For the test case of myoglobin (146 residues, 8 α-helices), this method produced a lowest energy conformation with an rms deviation of 6.2 Å from the crystal structure and which was in good qualitative agreement with the native topology.

This algorithm has been implemented on a parallel connection machine, and also takes advantage of some features of a massively parallel calculation, in particular the simultaneous generation of a large ensemble of results. Since the results of a simulated annealing calculation are necessarily statistical in nature it is important to have a large number of equivalent calculations to be able to evaluate the performance of the method. In addition, a genetic algorithm was used in which the different members of the ensemble were cut and reassembled as an additional means of generating new trial conformations. The generation of trial moves was accelerated by calculating in advance a list of trial loop segments and their corresponding geometries. Since the list was available to a larger number of structures, it resulted in a considerable time savings. Details of this procedure can be found in ref. 1.

However, subsequent results have shown that the simplified model fails to describe in sufficient detail the close-packing of residues found in native proteins.[2] For this reason, further refinement of the structures with the more detailed model using a more accurate potential is needed to generate results with a higher level of resolution than that obtained from the original hierarchical algorithm. In particular, a large part of the error is due to the flexible loop regions which are chosen randomly from an unbiased distribution. This article discusses the implementation of a refinement procedure which is designed to generate small changes in the structure and allow the determination of the loop regions to be improved. The refinement algorithm follows the same data-parallel construction as in the hierarchical case, with an additional benefit of distributing the storage requirements of a lookup table potential function. Since the advantage of using the reduced model is lost in the refinement procedure (at least in the present case) it is of great importance to obtain the best possible performance for the residue-based model. This study will explore the advantages and limitations of the parallel implementation and provide a comparison with a serial version of the same calculation.

Refinement Algorithm

The refinement algorithm is based on a model of the protein backbone in which each residue is described by the positions of six atoms: N, H, C_α, C_β, C, and O. The only degrees of freedom are the dihedral angles ϕ and ψ around the N—C_α and C_α–C bonds. The set of possible conformational states is restricted to a discrete set of points in the two-dimensional Ramachandran map

$$S_i = (\phi_i, \psi_i) \tag{1}$$

with the conformation of the molecule being represented simply as the set of states

$$C = \{S_1, S_2, S_3, \ldots, S_N\} \tag{2}$$

where N is the number of residues. The discrete states were chosen to represent observed conformations with sufficient variety to approximately describe any reasonable overall folded topology.[3]

The geometrical description of the molecule is simplified by the use of the same peptide geometry for each residue, making the conformation, C, sequence-independent (cis-proline is not considered in this model). Of course, the potential interactions between pairs of atoms are still dependent on amino-acid type.

SHORT SEGMENT REPLACEMENT

Because the states are discrete, a trial move consisting of changing the state of a single residue

would generate a rotation about that point which could lead to a large displacement at the ends of the molecule and a correspondingly low probability of acceptance. For this reason, the trial moves in ref. 1 consisted of the replacement of entire loops. In the present work, this strategy has been modified to allow the refinement of the existing loops in such a way as to make changes in the global topology that are within a specified range.

The shortest segment with the desired flexibility consists of three residues. This allows a single residue (in the center) to be moved, while the other two can undergo compensating rotations so that the net rotation of the ends of the molecule can be maintained close to its original value. Selecting the shortest such segment allows the loops to be evolved through a series of small changes which maintain as much as possible of the existing loop structure.

The basic trial move thus consists of the substitution of a set of three residues at a specified point

$$C' = \{S_1, \ldots, S'_i, S'_{i+1}, S'_{i+2}, \ldots, S_N\}. \quad (3)$$

The coordinates are updated by defining an absolute orientation, **U**, associated with each residue

$$\mathbf{U} = \begin{pmatrix} \hat{\mathbf{r}}_{N-C_\alpha} \\ \dfrac{\hat{\mathbf{r}}_{N-H} - \cos\theta\, \hat{\mathbf{r}}_{N-C_\alpha}}{\sin\theta} \\ \dfrac{\hat{\mathbf{r}}_{N-C_\alpha} \times \hat{\mathbf{r}}_{N-H}}{\sin\theta} \end{pmatrix} \quad (4)$$

where $\cos\theta = \hat{\mathbf{r}}_{N-C_\alpha} \cdot \hat{\mathbf{r}}_{N-H}$. A schematic illustration is shown in Figure 1 of the three basis vectors defined by the rows of **U**. The molecule is constructed by calculating the rotation, **T**, of each residue relative to its predecessor, which is an easily determined function of ϕ, ψ, and the constant bond angles of the peptide unit.[4] The cumulative rotation is thus

$$\mathbf{U}_{i+1} = \mathbf{U}_i\mathbf{T}_i = \mathbf{T}_1\mathbf{T}_2 \cdots \mathbf{T}_i \quad (5)$$

with $\mathbf{U}_1 = \mathbf{I}$. During this cumulative matrix multiplication, the positions of each atom in a residue along with the position of the following C_α can be determined relative to the current C_α center to generate a complete set of coordinates.

This procedure can be straightforwardly applied to carry out the subsequent segment replacements in the following manner. The new segment itself is generated by calculating a new **T'** for each new S' and using the fact that $\mathbf{U}'_i = \mathbf{U}_i$ to calculate

$$\mathbf{U}'_{i+3} = \mathbf{U}_i\mathbf{T}'_i\mathbf{T}'_{i+1}\mathbf{T}'_{i+2} \quad (6)$$

The net rotation of the remainder of the chain is equal to $\mathbf{U}^T_{i+3}\mathbf{U}'_{i+3}$ which can be applied with $\mathbf{r}_{C_\alpha, i+3}$ at the origin followed by a translation equal to $\mathbf{r}'_{C_\alpha, i+3}$. This method can also be trivially done in the opposite direction for the case where $i < N/2$.

The advantage of developing this standard description of the residue orientations is that the geometries of the segments can be described in a universal way in terms of the atomic coordinates without reference to the axes of the α-helices and β-strands as was the case with the original hierarchical algorithm. This also simplifies the process of generating a list of trial segments, since all trial moves will use segments of the same length and a three-residue piece can be substituted at any point in the chain. This use of a segment library has also been discussed previously in the context of genetic algorithms.[5]

The secondary structure can thus be omitted entirely from the algorithm if desired. However, it is still useful to think in terms of the original model of packing rigid cylinders into a compact shape. This can be achieved by simply identifying those residues which are to be allowed to change conformational state and those which are to be held fixed. This is a more general restriction, because any part of the molecule can be held fixed regardless of whether or not it actually has a regular structure. The algorithm can thus be used to selectively refine those parts of the structure which show the greatest variation. In the calculation, a label is assigned to each residue indicating whether or not it is to be fixed, and the sites for segment replacement trial moves are then chosen from among all possible sets of three consecutive

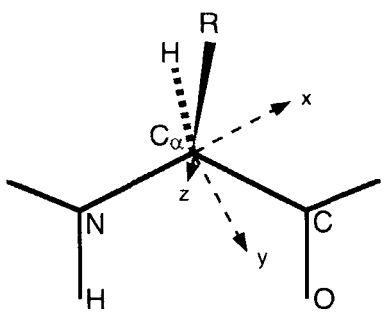

FIGURE 1. Schematic illustration of the orthogonal coordinate system used to define the absolute orientation of each C_α center. The *z*-axis is perpendicular to the plane of the page.

nonfixed residues. In a subsequent calculation the labels could also be reversed, which would allow for the selective refinement of the secondary structure elements using a set of conformational states and trial segments within a smaller region of the conformational space around the ideal α-helix and β-strand geometries.

With all trial segments being three residues, the number of distinct trial moves is N_d^3, where N_d is the number of discrete conformational states. Thus, N_d can be increased substantially from the value of 18 used in ref. 1 and still allow each potential trial move to be identified with a single integer label. The states were chosen to lie on alternate sites of a 5° square grid (648 points) restricted to the populated regions of the Ramachandran map, leaving 532 points. This greater number of conformational states allows the resolution of the model to be improved since the number of possible three-residue segments within a specified range of geometries will be comparable to that previously observed for entire long loops. However, rather than searching for a complete loop with an appropriate geometry, the present algorithm will allow for the successive refinement of an existing loop with partial substitutions while still maintaining a similar overall geometry. While the primary criterion for loop selection is still assumed to be the constraints of packing the helices, this refinement will allow the effects of the local interactions and the self-energy of the loop to be better taken into account.

SEGMENT LIST SORTING

When there is any sort of preselection of the segments, i.e., a test that can be used to reject possible segments prior to substituting them into the molecule, there is an advantage in compiling a precalculated list since the list can be used independently for a large number of structures in the ensemble and the calculation of unacceptable segments need not be repeated. Each structure will only select a small fraction of the available trial moves, but because there are many structures each segment in the list will on average be used several times. The number of possible segments, however, can be much larger than can be reasonably stored and therefore the list is constructed during the simulation and replaced after an appropriate number of trial moves.

To generate a list of segments which are similar in geometry to those already in the ensemble, a relatively strict screening of the random segments is carried out. This is done by assigning each segment to a particular "bin" biased on the relative rotation of its endpoints. The end-to-end distance is neglected in this step since it does not vary very much in a three-residue segment and a simple translation will have a small effect on the rest of the molecule relative to the rotation. The geometry of a segment is thus determined by five independent parameters, which are defined as

$$\begin{aligned} q_1 &= \cos^{-1}(\hat{\mathbf{x}}_1 \cdot \hat{\mathbf{r}}) \\ q_2 &= \cos^{-1}(\hat{\mathbf{x}}_2 \cdot \hat{\mathbf{r}}) \\ |q_3| &= \cos^{-1}\left(\frac{\hat{\mathbf{x}}_1 \cdot \hat{\mathbf{x}}_2 - (\hat{\mathbf{x}}_1 \cdot \hat{\mathbf{r}})(\hat{\mathbf{x}}_2 \cdot \hat{\mathbf{r}})}{\sin q_1 \sin q_2}\right) \\ |q_4| &= \cos^{-1}\left(\frac{\hat{\mathbf{y}}_1 \cdot \hat{\mathbf{r}}}{\sin q_1}\right) \\ |q_5| &= \cos^{-1}\left(\frac{\hat{\mathbf{y}}_2 \cdot \hat{\mathbf{r}}}{\sin q_2}\right) \end{aligned} \tag{7}$$

where $\{\hat{\mathbf{x}}_1, \hat{\mathbf{y}}_1, \hat{\mathbf{z}}_1\}$ are the rows of $\mathbf{U}_i$, $\{\hat{\mathbf{x}}_2, \hat{\mathbf{y}}_2, \hat{\mathbf{z}}_2\}$ are the rows of $\mathbf{U}_{i+3}$, and $\hat{\mathbf{r}}$ is the direction of the vector $\mathbf{r}_{C_\alpha, i+3} - \mathbf{r}_{C_\alpha, i}$. These coordinates have a range of $0 \le q_{1,2} \le \pi$ and $-\pi \le q_{3,4,5} \le \pi$ where the signs of the dihedrals $q_{3,4,5}$ are determined by the relation

$$\frac{q_i}{|q_i|} = \frac{s_i}{|s_i|} \tag{8}$$

where

$$\begin{aligned} s_3 &= \hat{\mathbf{r}} \cdot (\hat{\mathbf{x}}_1 \times \hat{\mathbf{x}}_2) \\ s_4 &= \hat{\mathbf{r}} \cdot \hat{\mathbf{z}}_1 \\ s_5 &= \hat{\mathbf{r}} \cdot \hat{\mathbf{z}}_2 \end{aligned} \tag{9}$$

are the corresponding triple products. These coordinates are shown schematically in Figure 2. The bins are defined by dividing this coordinate space up into equal intervals of π/n generating a total of $8n^5$ bins. The optimal choice of n depends on a number of factors such as the desired acceptance rate of trial moves and the diversity of the existing ensemble. For the purposes of this initial study, a value of $n = 6$ corresponding to a bin size of 30° was used.

The screening of random segments in the construction of the segment list is carried out by assigning each bin a weight based on its occupancy in the current ensemble. The geometries of all segments eligible for substitution in the ensemble are calculated and the number falling in each

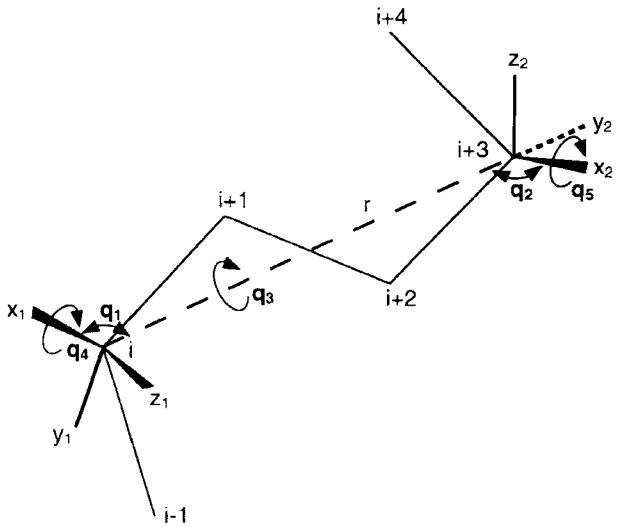

FIGURE 2. Schematic illustration of the angle parameters used to define the relative orientation of two C_α centers. The structure shown here is not intended to be realistic, since in general the C_α atoms would not be coplanar.

bin tabulated. These values are normalized so that the largest value is equal to 1.0 and the random segments are then added to the table with that probability. Note that this does not ensure a distribution of segments equal to that of the calculated weights, since there will also be a bias toward those geometries which are inherently more likely to occur as a function of the conformational states. However, the primary effect will be due to the fact that most bins will have a weight of zero and thus be excluded completely.

Once the list has been filled with the desired number of segments, it is sorted so that segments which are in the same bin are found consecutively. The location of each group of segments can thus be specified by the location of the first member of the group and the number in the group. The table lookup to select a trial segment from a particular bin then consists of picking a random number between 1 and the number of segments in the bin and counting from the starting location. For bins which are not found in the list, the lookup will be transfered to the neighboring bin (i.e., one which differs in only one of the five coordinates) which has the greatest number of segments. If there are no occupied neighbors, the lookup is unrestricted and consists of the entire list. This situation can arise during the simulation since the segments overlap and thus the replacement of a segment with one having a similar geometry may cause other segments having endpoints in its interior to be in bins not previously in the ensemble. Since the list is regenerated periodically with new weights, the list will evolve with the ensemble.

The selection of trial moves in the simulation consists of first determining the geometry of the segment to be replaced. The coordinates $\{q\}$ are determined from $\mathbf{U}_i$ and $\mathbf{U}_{i+3}$ to obtain the bin number. (The values of $\mathbf{U}_i$ and $\mathbf{U}_{i+3}$ are also needed to perform the coordinate rotation if the move is accepted.) A trial segment is chosen from the list for that bin based on the rules just described, and then constructed using the conformational states $\{S\}$ to give $\mathbf{U}'_{i+3}$. The trial move is then carried out and, if it is accepted, the molecular conformation is updated with the new segment $\{S'_i, S'_{i+1}, S'_{i+2}\}$.

LOOKUP TABLE POTENTIAL FUNCTION

The potential function used in these simulations has the form of a generalized distance-dependent contact potential. For each amino-acid pair there is a series of distance ranges, each of which has an energy associated with it which is derived from an analysis of the protein data bank. This was done to correct some of the deficiencies observed with the original continuous potential.[6] To further improve the precision, the potential table was extended to include four distances: $|\mathbf{r}_{C_\alpha,i} - \mathbf{r}_{C_\alpha,j}|$; $|\mathbf{r}_{C_\alpha,i} - \mathbf{r}_{C_\beta,j}|$; $|\mathbf{r}_{C_\beta,i} - \mathbf{r}_{C_\alpha,j}|$; and $|\mathbf{r}_{C_\beta,i} - \mathbf{r}_{C_\beta,j}|$. The use of 4 degrees of freedom is equivalent to considering not just the C_β—C_β distance, but also the relative orientations of the two C_α—C_β bond vectors. This corresponds to a simple approximation of the positions of the side chains relative to the backbone. For each distance there are 10 discretized values between 3 Å and 7.5 Å, yielding a possible total of 10^4 energy values for each amino-acid pair. The long range part of the potential was accounted for with a single-index table based on $|\mathbf{r}_{C_\beta,i} - \mathbf{r}_{C_\beta,j}|$, which extended to a distance of 27 Å. The details of this potential function and the calculation of the energies can be found in ref. 2. From a computational point of view, a lookup table potential of this nature has the advantage that the same format can be used as an approximation to any arbitrary function and, as more accurate approximations are developed, only the numbers will change.

Due to the geometrical constraints of the bonding and the excluded volume of the side chain atoms most of the possible combinations of the four indices will in fact never be observed. To

avoid having to store values for all possible combinations in the table, it would be useful to limit the table to those values which are observed in the protein data bank and to use a constant parameter (either zero or some positive "penalty") for all others. This can be done by treating the possible combinations of indices sequentially and at each step eliminating those choices which are no longer distinct. The initial lookup thus consists of a table of pointers with a default value for all combinations which have no nonzero entries for any choice of the remaining parameters, and unique labels for the remainder. Adding the next index then only requires a table of length $N \times$ (the number of distinct combinations), rather than N^3. The final step will thus be a pointer directly to the nonzero entries in the table. While this procedure would not be particularly useful for a random distribution of data, because of the geometrical relationship of the indices the potential table in this case has a banded structure making this a reasonable strategy. In addition, the further simplifying assumption is made that the structure of the potential table will be the same for each amino-acid pair. Since the side chains are of different sizes, this requires the inclusion of a number of zero values in the potential table, but at the same time reduces the size of the pointer arrays by a substantial fraction. The pointer arrays in this case were generated so as to preserve a unique label for any combination of the distance indices which had a nonzero entry for at least one amino-acid pair. In the implementation used here, there are a total of 25 indices for each distance (including a long range extension), yielding a total number of addressable energies of 8.20×10^7. The procedure described here used pointer arrays with a combined length of 1.87×10^4 and a potential table of length 4.36×10^5 to represent 5.30×10^4 nonzero entries.

Implementation on a Parallel Computer

COMPUTATIONAL MODEL

The principal models of parallel execution generally fall between two limits, data parallelism and control parallelism, which are in turn related to the limiting data structures of distributed and shared memory. In a data-parallel distributed-memory model, all processors carry out the same instructions, each acting on its own piece of the data. In a multiple-instruction shared-memory model on the other hand, each processor has access to the same data and carries out a different operation on them. Many possible combinations of these methods are also possible, along with many ways of managing communication among the processors.

The present algorithm was implemented on a parallel Connection Machine (CM-5) which makes use of elements of both data-parallel and control-parallel structures. The machine consists of a partition manager (PM) which controls execution, and a number of nodes which control communication and memory access. Each node in turn consists of four vector processors (VUs) each of which has its own dedicated memory. Execution can be carried out in either a data-parallel model in which each node carries out the same instructions and communication is invoked automatically for array operations which span more than one VU, or in a local-node model in which each node carries out different instructions while retaining a data-parallel structure at the level of the VUs. Communication in the latter case can be handled through the use of explicit message passing between nodes, or by using a global program to move data and call subroutines which then execute locally on the nodes. The optimum performance on the CM-5 is obtained in the VUs, with the major bottlenecks coming from internode communication. The VU performance is further dependent on the vector length of each array operation thus requiring further data parallelism within each VU.

There are three important ways in which parallelism can be exploited in the hierarchical algorithm. The first and most obvious is at the data-parallel level where the distinct structures in the ensemble are distributed across the nodes and the trial moves are carried out independently. Relative performance can be improved in this case simply by increasing the size of the ensemble, and virtually no communication is required. This structure can also be adapted to include the use of a genetic algorithm as was done in ref. 1, but that aspect will not be addressed here. There are, however, two instances where this simple approach is limited, and other types of parallelism need to be introduced. The first concerns the segment list, for which the advantage lies in the number of independent structures making use of the same list. What is in principle a globally accessible array must therefore be distributed for use in data-parallel trial moves. The second concerns the calculation of interresidue distances where the data-parallel structure can never overcome the N^2 scaling of the distance matrix and thus reduce its cost rela-

tive to the other parts of the program. This must be addressed by invoking a multiple-instruction model in which different nodes are allowed to evaluate different parts of the distance matrix of the same structure. Finally, these issues are coupled in the case of the potential table. The larger size of the table means that it should be distributed across the nodes rather than copied in its entirety to each one. That, however, requires having different nodes calculating different contributions to the energy of each structure. The major difficulties in the current implementation are thus related to combining these different models into a single program and trying to avoid excessive internode communication in moving from one to another.

DISTRIBUTED SEGMENT LIST

An important requirement of the segment list is that the selection of trial moves be carried out locally without any interprocessor communication, thus requiring that each VU have its own list, and that the lookup for each molecule be restricted, therefore, to those segments located on the same VU. On the other hand, the screening of the segments is most easily done globally, since the generation of segments is completely independent, and it is only the total number of segments in the list that is important. The list is then sorted to have the same pattern of bins and hence the same index arrays on each VU. This allows the sorting to be done within a data-parallel routine and the segments in each bin to be evenly redistributed across the machine regardless of where they are originally constructed. Periodically during the calculation the entire segment list is circularly shifted across the VUs so that each molecule will see each local list before the entire array is recalculated. The sorting procedure is therefore carried out so as to ensure that each position in the list is in fact a set of segments in a single bin with a different member of the set in an equivalent location on each VU. However, this requires that the sorting involve communication between nodes both to move the segments and to calculate the total numbers of each type. An alternative would be to generate and sort the list on each node independently; however, this would restrict each list to the segments generated in place meaning that some nodes would take longer than others. More importantly though, the lookup is parallelized at the VU level, whereas the sorting can be done at most at the node level, requiring either the same pattern on the VUs of a single node or the sorting of the lists in each VU of a node sequentially. Because of this additional complexity this was not pursued further.

The list is generated by filling an array of segments with random conformational states and calculating the geometries and corresponding bins of each. In each case, a random number is compared to the weight associated with each bin and those segments for which the weight is greater are accepted. The remaining segments are replaced with new random conformations and the procedure is repeated until the desired number of segments has been found. Since the search is restricted to positions in the array that have not yet been filled, the array used is several times larger than the desired segment list. This also allows the time-consuming operations of constructing the segments and calculating their geometries to be carried out with the largest possible vector length.

Since it is necessary to have a number of segments in each bin that is a multiple of the number of VUs, the array was overfilled until this condition was met. This required counting the number of accepted segments in each bin, which is an expensive operation because the number of bins can be very large. This was partially alleviated by having a crossover point after which only bins that were occupied were considered and all others were given a weight of zero and no longer needed to be counted. The final segments comprising the set for each occupied bin were redistributed across the VUs using the intrinsic function `PACK` and then placed in the correct position locally on each VU according to the index arrays which will be known at this point in the routine.

DISTRIBUTION OF PAIRWISE INTERACTIONS

The $\mathcal{O}(N^2)$ calculation of the pairwise interactions represents the most CPU-intensive part of the algorithm. To maximize the vector length for this routine, the list of (i, j) pairs of residues is divided into blocks. The distance indices for each block are accumulated and the subsequent table lookups are then carried out simultaneously for the entire block. The same strategy however is not effective for the distances themselves as it would involve accumulating the cartesian coordinates of each atom rather than single integers and the overhead from this step would be greater than the potential gain. For this reason, the calculation is further distributed by assigning to each node a different block of pairs. This means that the optimal structure for the energy evaluation would be one where

each node would have a copy of all of the coordinates but only a portion of the potential table and distance matrix. Unfortunately, this is the inverse of the data-parallel structure of the trial moves and thus requires communication to make the coordinates of the structures on each node available to the other nodes. An alternative would be to have copies of all coordinates permanently on each node, and distribute the calculation of the trial moves. This would, however, require the updated coordinates to be sent from the node carrying out the move to all of the others and would amount to the same thing.

Global / Local Programming

The distance calculation is carried out in an explicit loop over the index pairs using the fast indirect addressing available within a single VU to access the cartesian coordinates of the atoms. By assigning to each node a different set of pairs the number of iterations of this loop was reduced by a factor of the number of nodes while the vector length of the calculation was increased by the same amount by having each node work with the entire ensemble. This rearrangement of the calculation led to improved performance, but it was partially offset by the communication required to send a copy of each structure to each of the other nodes so that each would have a access to the complete ensemble. This is done by performing a circular shift of the coordinate array using the intrinsic function `CSHIFT` followed by calling a local-node routine to copy the data to the appropriate location in an expanded array aligned on each VU. A local-node routine is used to do the distance calculation and accumulate the distance indices, so that each node can use different loop indices determined in a precalculated array and selected based on the processor ID. Following the table lookups to determine the potential, the energies for each molecule are summed across the nodes using a similar combination of `CSHIFT` and a local-node routine.

Allocation of Potential Table

The lookup table containing the potential function requires the storage of a large amount of data. In this example, as described earlier, it is on the order of 2 Mb, which must be stored on each VU to avoid any interprocessor communication during the calculation. Since the available memory on each VU is 8 Mb this represents a substantial cost and presents a barrier to the development of more highly resolved potential tables with additional data. This problem can be overcome with the parallel structure described above by assigning the amino-acid pairs to each of the nodes in such way that each node only requires a portion of the potential. This requires a compromise between the even distribution of the data and the even distribution of the computational load. This means that, in practice, the potential will be divided into sections that consist of several amino-acid pairs, but a significant savings can still be achieved.

With the present example of myoglobin, there are 10,440 pair interactions which can be divided across 16 nodes with the number of pairs per node ranging from 626 to 668 with the corresponding number of amino-acid potential functions ranging from 6 to 16. This represents a savings in memory by a factor of 13 with an overhead of only 2.3%. On a large enough partition, the memory requirement could be reduced by a factor of 210 (i.e., each node could be assigned a single amino-acid pair). This would allow the storage of a potential consisting of over 400 Mb of data in the same fraction of VU memory currently used.

Results

Results are discussed here for a test calculation involving the refinement of the myoglobin structure previously obtained in ref. 1. This structure was found to be essentially equal in energy to the native crystal structure with the original potential function. However, the table-based screening potential developed in ref. 2 re-established a substantial energy gap between this structure and the native one, as shown in Table I. The refinement was carried out by generating an ensemble of 128

TABLE I.
Comparison of Energy Minimization Results.

	Ref. 1 pot. (arbitrary units)	Ref. 2 pot. (arbitrary units)
Initial ensemble average[a]	−5.57	9642
Initial low-E structure	−10.45	9315
Final ensemble average	[b]	8705
Final low-E structure		8332
Native structure	−10.49	8327

[a] From ref. 1.
[b] The ref. 1 potential was not implemented in the refinement program.

copies of the low-energy structure and performing a simultaneous simulated annealing minimization using the algorithm presented here. The simulation was run approximately for 36 h, comparable to the approximately 48 h used to generate the results in ref. 1. As shown in Table I, the refinement was successful in eliminating the energy gap and producing a new structure comparable in energy to the native structure based on the new potential. However, the rms deviation from the native was virtually unchanged, indicating that, while the local energy of the structure was improved, the potential function is still not sufficient to narrow the nativelike basin of attraction. More work is currently underway to evaluate other potential functions in an attempt to address this problem.

The improved spatial resolution of the refinement algorithm can be shown independently by comparing the results of minimizations carried out using the rms deviation itself as a potential function. The original hierarchical algorithm was limited to approximately 5 Å, whereas the refinement algorithm reduced that to approximately 3 Å. Direct fitting of the crystal structure suggests that the limit imposed by the use of ideal peptide geometries and discrete conformational states is on the order of 1.5 Å.

COMPARISON TO SERIAL ALGORITHM

The performance comparisons were carried out using a shortened version of the calculation which performed fewer iterations but was otherwise identical. These results are for an ensemble of 128 molecules, 81,920 trial segments in the list, and 5120 trial substitutions per molecule. The results are shown in Table II. The serial version was written in standard F77 and was identical on the IBM and Silicon Graphics machines. The load distribution between the different parts of the program was nearly the same on the serial machines, but differed significantly on the Connection Machine. The serial segment list generation was somewhat simpler, since the random segments were generated one at a time. This eliminated the overhead associated with calculating the whole array at once and carrying the already accepted segments, as well as the overfilling required to reproduce the same pattern of bins on each node. The total number of calculations of segment coordinates in the serial program was 3.60×10^7, whereas for the parallel program this number was 5.51×10^7. This could be partially reduced by running the calculation independenty on each node, however, it still would not account for the difference in relative speeds. The parallel program spends about 50% of the list generation time calculating the segment coordinates and geometries, whereas in the serial version this is about 80%, suggesting that the bottleneck lies elsewhere. The potential evaluation routine seems to show the best parallel performance, while the remainder of the program shows evidence of further bottlenecks consuming a much greater fraction of the time than in the serial case. The algorithmic differences caused a negligible difference in the results of the minimization.

TABLE II.
Timing Results for Parallel and Serial Implementations.

	Total	Pot. eval.		List gen.	
	[s]	[s]	(%)	[s]	(%)
16-node CM-5	5501	2285	(41.5)	1901	(34.6)
SGI Indigo[a]	28,930	23,900	(82.6)	4123	(14.3)
IBM RS6000[b]	7962	6557	(82.4)	1064	(13.4)

[a]100 MHz R4000 CPU.
[b]67 MHz POWER2 CPU.

PERFORMANCE SCALING WITH ENSEMBLE SIZE

To obtain the best performance from the parallel program, the problem should be set up so as to take the most advantage of the types of parallelism being used. Within the data-parallel model, this corresponds to increasing the number of independent data elements undergoing each operation. This is particularly important on the CM-5, where the underlying hardware performance is also dependent on the data-parallel structure at the level of each individual VU, namely the vector length.

The vector length for the table lookups can be made large by increasing the number of distance pairs accumulated during each iteration of the local-node distance routine. The value of 128 used here seems to be sufficient, since changing it to 256 had no significant effect on the performance. For the distance calculation, however, the only way to increase the vector length is to increase the size of the ensemble. The results are shown in Table III for three different sizes, with other parameters remaining constant. As expected, the performance of the potential calculation is poorer with fewer molecules, however, the results for a larger ensemble do not show any improvement. It is not clear

why this is the case, but suggests that this simple model breaks down at this point and that the net performance is not trivially improved simply by adding more structures. Since the vector length in the segment list generation routine depends on the length of the list, it is already near the performance limit. This is shown in Table III where the list length was scaled in proportion, i.e., holding the number of segments per molecule constant. There is little change between the smaller ensembles, while the performance was degraded in the larger case. This was due to an increase in the overhead (described earlier) due to memory limitations. In this case, a smaller list could also have been used with more frequent updates.

PERFORMANCE SCALING WITH PARTITION SIZE

The underlying objective of parallel programming is to set up the problem so that the total time of the calculation can be proportionally reduced by adding more processors. In the present case, the performance of the program on different-sized partitions was not explicitly tested, but can be estimated from the results here. The primary advantage of a larger partition would be in the segment list generation. The time required would still depend on the number of segments per VU in the list; however, because the segments on each VU are distinct the overall list would be larger along with the number of trial moves performed while it is cycled through the nodes. This means that, in effect, it needs to be updated less often, and therefore takes up a smaller proportion of the total time. This is just a consequence of the overall economy of scale arising from the use of a list of segments each of which can be used in a number of independent trial moves.

For the potential evaluation, the situation is more complex. The vector length is increased by sending to each node a copy of all coordinates and assigning to each node a different set of residue pairs to calculate. This increase in vector length could thus in principle be equal to the number of nodes. However, the memory requirements of this procedure would rapidly become unmanageable. More importantly, the internode communication required to transfer the coordinates is also proportional to the number of nodes. In practice, it is necessary to explicitly divide the nodes into a number of independent blocks to achieve a reasonable balance. The results of different divisions of a 16-node partition are shown in Table IV. In going from eight blocks of two nodes to four blocks of four nodes, there is virtually no change in overall performance. The gain in speed due to the increased vector length is offset by the added communication time. In the 16×1 case (not shown), any reduction in the time for the distance matrix is more than canceled by an additional doubling of the communication requirement. In the 2×8 case, while the communication is small the loss in vector performance is substantial. Best performance is therefore achieved with approximately eight independent blocks, and with this number held constant the performance would be expected to be similar on a larger partition with the same number of molecules per node. The balance might, however, be different for other relative ensemble sizes. (The results in the previous subsection used a 4×4 distribution.)

Conclusion

Preliminary results have been shown which indicate that the short-segment replacement algorithm is effective in minimizing the energy of a reduced model of a protein. This is a result of a two-step procedure in which trial segments are screened independently with a simple geometric constant based on the end-to-end rotation introduced by the segment. The constraint is deter-

TABLE III.
Comparison of Results As a Function of Vector Length. Relative Values Are with Respect to the 128-Member Ensemble and Normalized by the Number of Structures.

	Total		Pot. eval.		List gen.	
Ens. size	[s]	(rel.)	[s]	(rel.)	[s]	(rel.)
64	4076	(1.48)	2021	(1.77)	1017	(1.07)
128	5501	(1.0)	2285	(1.0)	1901	(1.0)
192	8581	(1.04)	3508	(1.02)	3482	(1.22)

TABLE IV.
Timing Results for Different Data Distribution Schemes.

Partition struct.	Pot. eval. (total) [s]	Broadcast [s]	Distance calc. [s]
8 × 2 nodes	2326	659	950
4 × 4 nodes	2371	386	1277
2 × 8 nodes	3635	167	2277

mined by the distribution of segment geometries in a current ensemble so as to produce a list of screened segments with a similar distribution. Since the resulting trial moves are known to cause limited perturbations in the overall structure, this leads to a higher acceptance rate and increased efficiency in a simulated annealing calculation which selects moves from this list.

This has been implemented on a parallel computer, with mixed results. An important advantage lies in the ability to store very large amounts of data, which is important for the use of a table lookup potential function. Although this is not critical in the current implementation, it allows for the development of more detailed and precise tables requiring more memory. This is anticipated, since the current potential, after minimization, did not seem to provide sufficient selectivity to generate a more accurate model of the native structure.

A comparison was made to a serial implementation of the same algorithm which showed that the balance of computational effort varies significantly between the two. While some parts of the program appear to be well parallelized, the overall performance is limited by interprocessor communication which is necessary in combining different parallelization models in the algorithm. The generation of the segment list, in particular, consumes a proportionally larger fraction of the total time, although this effect can be reduced by increasing the size of the partition.

The relatively good performance of the program on a single processor suggests that the algorithm is rather flexible, and the fact that similar results were obtained on different serial platforms without modification suggests that the code is reasonably portable. This offers the possibility that the parallel implementation might be improved by specifically addressing the current limitations. Although this algorithm has not yet been implemented on any other parallel platform, some properties of the CM-5 can be used to speculate on its general applicability. The performance of the CM-5 depends heavily on the vector processing on the VUs, which imposes an additional constraint on the program. On a machine with better scalar performance the algorithm would be more flexible and the data parallelism could be exploited at the processor level. The current implementation also relies on a single-instruction model of interprocessor communication, whereas a message-passing environment might be more flexible in allowing each processor to determine individually what data it needs to transfer. This could be used to reduce the cost of copying coordinates between the trial move and energy evaluation parts of the program. The strength of the CM-5 is in those parts of the program which conform well to the data-parallel model and use synchronized communication. On a different platform there would likely be a trade-off in performance between routines which would depend on the speed and structure of the communication network; however, the current program is sufficiently flexible that the load balance could be adjusted accordingly, as was the case for the different data distribution schemes discussed earlier.

There are several aspects of the algorithm itself that also need to be studied further, including the optimal size of the segment geometry bins, the number of bins to include in the list, and various parameters associated with the minimization. However, since these results depend substantially on the nature of the ensemble and the level of refinement being carried out, this work is being carried out in conjunction with the development of improved potential functions, which are needed to better evaluate the performance of the method. Although this will vary the computational demands of different parts of the program, the present study is an important step in establishing the feasibility of such an approach.

Acknowledgments

Most of the work described here was carried out the Columbia University Center for Biomolecular Simulation with the support of the National Institutes of Health through Grant #P41-RR06892 to R.A.F. Financial support was also provided to J.R.G. by the Université de Montréal and CERCA. Assistance with several technical aspects of the parallel implementation is also gratefully acknowl-

edged from the support staff at Thinking Machines Corporation, and Chris Marshall in particular.

References

1. J. R. Gunn, A. Monge, R. A. Friesner, and C. H. Marshall, *J. Phys. Chem.*, **98,** 702 (1994).
2. A. Monge, E. J. P. Lathrop, J. R. Gunn, P. S. Shenkin, and R. A. Friesner, *J. Mol. Biol.*, **247,** 995 (1995).
3. M. J. Rooman, J. A. Kocher, and S. J. Wodak, *J. Mol. Biol.*, **221,** 961 (1991).
4. S. D. Stellman and P. J. Gans, *Macromolecules*, **5,** 516 (1971).
5. S. Sun, *Prot. Sci.*, **2,** 762 (1993).
6. G. Casari and M. J. Sippl, *J. Mol. Biol.*, **224,** 725 (1992).